**SVa: Secure Virtual Architecture**

http://sva.cs.uiuc.edu

John Criswell, Andrew Lenharth, Dinakar Dhurjati, Nicolas Geoffray, Vikram Adve

---

### Secure Virtual Architecture

- **Commodity ISA**
  - Virtual ISA
  - Native ISA

- **Hardware**
  - Compiler-based virtual machine
  - Uses analysis & transformation techniques from compilers
  - Supports commodity operating systems (e.g., Linux)

- **Software Flow**
  - Rich virtual instruction set enables sophisticated program analysis
  - Type restrictions enable safer verifier
  - Provides safe execution environment for commodity OSs
  - Compiler + small privileged run-time
  - Can resolve novel situations in security problems

---

### SVA Safety Guarantees

<table>
<thead>
<tr>
<th>Safe Language</th>
<th>SVA Secure Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array indexing within bounds</td>
<td>Array indexing within bounds</td>
</tr>
<tr>
<td>No uses of uninitialized variables</td>
<td>No uses of initialized variables</td>
</tr>
<tr>
<td>Type safety for all objects</td>
<td>Type safety for initialized objects</td>
</tr>
<tr>
<td>No uses of changing pointers</td>
<td>Pointer checks prevent changes to heap objects</td>
</tr>
<tr>
<td>Control flow integrity</td>
<td>Control flow integrity</td>
</tr>
<tr>
<td>Sound operational semantics</td>
<td>Sound operational semantics</td>
</tr>
</tbody>
</table>

- **Dangling pointers & non-type-safe objects do not compromise other guarantees**
- **Novel bounds checking is compatible with external components**
- **Similar safety guarantees for applications, kernel**

---

### Improved Object Lookups

- **Pointers**
  - Use two-level analysis (ISA) and Automatic Pool Analysis to group objects into logical partitions
  - Use machine word objects in partitions
  - Run-time checks only on objects in a single partition
  - Reduced slowdown from 44% to 12% (average 12%) for nearly all standalone programs, cleaners

- **SVM Memory Safety**
  - Insert object registration at allocation
  - Insert bounds check when the compiler cannot prove it safe

---

### Linux Kernel Modifications

- **SVA-ISA**
  - OS-level instructions support commodity OSs
  - Removes difficulty to analyze assembly code
  - Encourages privileged operations
  - Like porting to a new hardware architecture

- **SVA-OS**
  - OS-neutral instructions support commodity OSs

---

### Exploits

- **Tried 5 memory exploits that work on Linux 2.4.22**
  - Caught 4 out of 5 exploits
  - Uncaught exploit due to code not instrumented with checks

---

### Future Work: Execution Environment

- **Enhanced Memory Safety**
  - Prevent MMU changes from violating safety guarantees
  - Prevent safety violations from external I/O operations
  - Ensure state manipulation performed by OS is safe

- **Cross-System Analysis**
  - Faulty OS cannot violate application memory safety
  - Extend safety verifier to handle other security properties
  - Properties annotated in type system can be compact, efficient
  - More general proofs e.g., in Proof Carrying Code possible

---

### Future Work: Recovery

- **Provide semantic & support to recover OS from unexpected errors**
  - Don’t convert memory safety errors into Denial-of-Service issues
  - Errors may happen at arbitrary places in the kernel
  - Check invariants for security policy enforcement
  - ECC errors on memory operations
  - Pointer checks prevent changes to heap objects
  - Leverage existing error return semantics to provide recovery
  - OS has memory safety
  - Develop techniques to transition from unexpected errors to existing error handler
  - Preserve the kernel in a safer (a.k.a. correct) state when error occurs

---

### Future Work: Novel Applications

- **Treated Logging**
  - Virtual machine logs all OS operations security
  - Complete traces for operators within the kernel
  - Track changes to sensitive kernel application code

- **Encourage information flow policies uniformly within applications and kernel code**

- **Encourage cross-application kernel boundary enforcement**

- **Encourage cross-application kernel boundary enforcement (CAME)**
  - Encourage for commodity OS kernel code (e.g., Linux, Mac OS X)

- **Application Policy**
  - Policy provide application memory from compromised OS

- **Automatically encrypt private data through OS for I/O**

---

### Software Flow

- **Software Flow**
  - Machine checks only check objects in a single partition
  - Ensure state manipulation performed by OS is safe
  - Uses analysis & transformation techniques from compilers
  - More general proofs (e.g., as in Proof Carrying Code) possible

---

### Virtual Instruction Set

- **SVA-CPU**
  - Based on LLVM™
  - Type Explicit Control Flow Graph, Explicit SSA form
  - Sophisticated compiler analysis and transformation
  - Like porting to a new hardware architecture

### SVM Metadata:

http://sva.cs.uiuc.edu

---

### Performance

- **Web server bandwidth**
  - Small impact on local applications except ld.so (extremely system-call intensive)
  - Many short data transfers suffer high memory safety overhead

- **Server latency overhead**
  - Overhead acceptable for bulk data transfers

---

### Future Work

- **Secure Virtual Architecture: A Safe Execution Environment for Commodities Operating Systems**
  - John Criswell, Andrew Lenharth, Dinakar Dhurjati, and Vikram Adve
  - Awarded an Audience Choice Award

---

**University of Illinois at Urbana-Champaign**

www.iti.uiuc.edu